TP N°4 : Algorithme de Roy-Floyd-Warshall

1 Justifier que s'il existe un chemin de poids minimal entre deux sommets quelconques, alors il n'y a pas de cycle de poids strictement négatif dans G. Par la suite, on considérera que G n'admet pas de cycle de poids négatif.

> **Correction :** S'il existait un cycle de poids d strictement négatif dans G, alors en le parcourant n fois, on abaisserait le poids du chemin de $n \times d$, or
>
> $$n \times d \longrightarrow -\infty$$
>
> il suffirait de le parcourir suffisamment de fois pour que ce chemin soit plus plus petit que le chemin de poids minimal, ce qui est absurde. Il ne peut exister de chemin de taille minimal.

2 Justifier que si $p = \left((i_0, i_1, w_{i,i_1}), \ldots, (i_{\ell-1}, j, w_{i_{\ell-1},j})\right)$ est un chemin de poids minimal entre les sommets i et j, alors pour tout sommet intermédiaire k, par lequel passe le chemin p, les chemins partiels de i à k et de k à j sont eux-mêmes minimaux.

> **Correction :** Encore par l'absurde, s'il existait un chemin de poids strictement plus petit entre i et k (ou respectivement, entre k et j), il suffirait d'utiliser ce chemin entre i et k puis le chemin partiel de p entre k et j pour diminuer strictement le poids de p, ce qui est absurde car p est supposé minimal.

Principe

On peut définir, dans $\mathcal{C}_{i,j}$, une suite $\left(\mathcal{C}_{i,j,k}\right)_{k\geq 1}$ de sous-ensembles (croissant dans le sens de l'inclusion) où chaque $\mathcal{C}_{i,j,k}$ est formé des chemins de i à j dont les sommets intermédiaires, s'ils existent, sont dans $[\![0, k-1]\!]$. On notera $\mathcal{C}_{i,j,0}$ l'ensemble (éventuellement vide) des arcs reliant i à j. On observe alors que si p est un chemin de poids minimal dans $\mathcal{C}_{i,j,k}$,

- soit il "passe" par $k-1$ et p est la concaténation de deux chemins, de i à $k-1$ et de $k-1$ à j qui sont de poids minimaux et respectivement dans $\mathcal{C}_{i,k-1,k-1}$ et dans $\mathcal{C}_{k-1,j,k-1}$
- soit il ne passe pas par $k-1$ et alors c'est aussi un chemin de poids minimal entre i et j dans $\mathcal{C}_{i,j,k-1}$

3 Que représente $\mathcal{C}_{i,j,n}$?

> **Correction :** Il représente, par définition, tous les chemins menant de i à j et utilisant les sommet de $[\![0, n-1]\!] = S$, donc
>
> $$\mathcal{C}_{i,j,n} = \mathcal{C}_{i,j}$$

Construction de l'algorithme

4 Écrire une fonction python `matrix_adj(D)` qui prend en argument un dictionnaire d'adjacence D d'un graphe et renvoie la matrice d'adjacence associée. On pourra convenir que l'absence d'arc ou de chemin entre i et j est signalée par $W_{i,j} = n * \max\limits_{(u,v)\in S^2} (|w_{u,v}|) + 1$, puisqu'aucun chemin sans cycle n'atteindra ce poids.

Correction :

```
def matrix_adj(D):
    ''' Renvoie la matrice d'adjacence du graphe orienté et valué.
    D : dictionnaire contenant les triplets de A dont la première composante sert de clé'''
    n=len(D)
    m=0
    for i in D:
        for k in D[i]:
            if k[1]>m:
                m=k[1]
    W=np.zeros([n,n])+n*m+1
    for i in D:
        for k in D[i]:
            W[i][k[0]]=k[1]
    return W
```

On note $\ell(i,j,k)$ le poids minimal d'un chemin de i à j qui appartient à $\mathcal{C}_{i,j,k}$. Ce poids est $+\infty$ si cet ensemble est vide

5 Que vaut $\ell(i,j,0)$? Donner une relation de récurrence reliant $\ell(i,j,k)$ et les éléments de la forme $\ell(u,v,k-1)$ avec $(u,v) \in S^2$ en utilisant le principe de l'algorithme RFW.

Correction : $C(i,j,0)$ est l'ensemble des arcs reliant i à j (pas de sommet intermédiaire). On a donc bien $\ell(i,j,0) = W_{i,j}$. D'après le principe, si un chemin minimal de $C(i,j,k)$ ne passe pas par $k-1$ il est dans $C(i,j,k-1) \subset C(i,j,k)$ et reste donc minimal. Dans ce cas, $\ell(i,j,k) = \ell(i,j,k-1)$. Sinon, sa longueur est la somme des longueurs de deux chemins minimaux de $\mathcal{C}_{i,k-1,k-1}$ et $\mathcal{C}_{k-1,j,k-1}$. Dans ce cas, $\ell(i,j,k) = \ell(i,k-1,k-1) + \ell(k-1,j,k-1)$ La formule de récurrence est :

$$\ell(i,j,k) = \max\left(\ell(i,j,k-1), \ell(i,k-1,k-1) + \ell(k-1,j,k-1)\right)$$

6 Écrire une fonction `RFW(W)` qui prend en argument la matrice d'adjacence d'un graphe valué et renvoie la matrice contenant les longueurs des chemins de poids minimaux.

Correction :

```
def RFW(W):
    """
    W: matrice d'adjacence du graphe, tableau carré de taille le nombre de sommets
    Renvoie le tableau des poids des chemins de poids minimal
    """
    n,m=W.shape
    mon_inf=max( [ max( [W[i][j] for i in range(n)]) for j in range(m) ] )
    L=[W]
    for k in range(1,n+1):
        D_loc=L[-1]
        D=np.zeros([n,n], dtype = float)
        for i in range(n) :
            for j in range(n) :
                D[i][j]=min(D_loc[i][j], D_loc[i][k-1]+D_loc[k-1][j], mon_inf)
        L.append(D)
    return D_loc
```

7 Estimer la complexité en temps et en espace de cette algorithme en fonction du nombre de sommet n.

Correction : La complexité en temps est clairement $O\left(n^3\right)$. L'espace mémoire occupé est en $O\left(n^3\right)$ également. Il n'est toutefois pas utile de garder toutes les matrices et le script ci-dessous est en $O\left(n^2\right)$ pour ce qui est de la mémoire occupée.

8 Modifier la fonction précédente en `RFW2(W)` qui prend en argument la matrice d'adjacence d'un graphe valué et renvoie la matrice précédente et un conteneur qui donne pour chaque couple (i, j) le chemin de poids minimal et le poids associé.

Correction :

```
def RFW2(W):
    """
    W: matrice d'adjacence du graphe, tableau carré de taille le nombre de sommets
    Renvoie le tableau des poids des chemins de poids minimal et la liste de ces chemins
    """
    n,m=W.shape
    mon_inf=max( [ max( [W[i][j] for i in range(n)]) for j in range(m) ] )
    L=[W]
    CC=[[[ [i,j] if W[i][j]<mon_inf else [] for j in range(n)] for i in range(m)]]
    for k in range(1,n+1):
        D_loc=L[-1]
        C_loc=CC[-1]
        D=np.zeros([n,n], dtype = float)
        C=C_loc
        for i in range(n) :
            for j in range(n) :
                a=min(D_loc[i][j], D_loc[i][k-1]+D_loc[k-1][j], mon_inf)
                D[i][j]=a
                if a==D_loc[i][k-1]+D_loc[k-1][j]:
                    C[i][j]= C_loc[i][k-1][:-1]+C_loc[k-1][j]
        L.append(D)
        CC.append(C)
    return D_loc, C_loc
```

9 Tester avec le graphe suivant :

Correction :

```
D={0:[[1,1],[2,1]], 1:[[6,3],[4,1]],2:[[3,2]],3:[[1,1],[5,1]],4:[[7,1],[9,3]] \
,5:[[6,1]],6:[[7,2]],7:[[8,1]],8:[[4,1],[9,1]],9:[[10,2]],10:[]}
W=matrix_adj(D)
D,C=RFW2(W)
print(D)
print(C)
```

cela renvoie :

```
[[ 34.   1.   1.   3.   2.   4.   4.   3.   4.   5.   7.]
 [ 34.  34.  34.  34.   1.  34.   3.   2.   3.   4.   6.]
 [ 34.   3.  34.   2.   4.   3.   4.   5.   6.   7.   9.]
 [ 34.   1.  34.  34.   2.   1.   2.   3.   4.   5.   7.]
 [ 34.  34.  34.  34.   3.  34.  34.   1.   2.   3.   5.]
 [ 34.  34.  34.  34.   5.  34.   1.   3.   4.   5.   7.]
 [ 34.  34.  34.  34.   4.  34.  34.   2.   3.   4.   6.]
 [ 34.  34.  34.  34.   2.  34.  34.   3.   1.   2.   4.]
 [ 34.  34.  34.  34.   1.  34.  34.   2.   3.   1.   3.]
```

```
 [ 34.  34.  34.  34.  34.  34.  34.  34.  34.  34.   2.]
 [ 34.  34.  34.  34.  34.  34.  34.  34.  34.  34.  34.]]



[[[], [0, 1], [0, 2], [0, 2, 3], [0, 1, 4], [0, 2, 3, 5], [0, 1, 6], [0, 1, 4, 7
], [0, 1, 4, 7, 8], [0, 1, 4, 7, 8, 9], [0, 1, 4, 7, 8, 9, 10]], [[], [], [], []
, [1, 4], [], [1, 6], [1, 4, 7], [1, 4, 7, 8], [1, 4, 7, 8, 9], [1, 4, 7, 8, 9,
10]], [[], [2, 3, 1], [], [2, 3], [2, 3, 1, 4], [2, 3, 5], [2, 3, 5, 6], [2, 3,
1, 4, 7], [2, 3, 1, 4, 7, 8], [2, 3, 1, 4, 7, 8, 9], [2, 3, 1, 4, 7, 8, 9, 10]],
 [[], [3, 1], [], [], [3, 1, 4], [3, 5], [3, 5, 6], [3, 1, 4, 7], [3, 1, 4, 7, 8
], [3, 1, 4, 7, 8, 9], [3, 1, 4, 7, 8, 9, 10]], [[], [], [], [], [4, 7, 8, 4], [
], [], [4, 7], [4, 7, 8], [4, 7, 8, 9], [4, 7, 8, 9, 10]], [[], [], [], [], [5,
6, 7, 8, 4], [], [5, 6], [5, 6, 7], [5, 6, 7, 8], [5, 6, 7, 8, 9], [5, 6, 7, 8,
9, 10]], [[], [], [], [], [6, 7, 8, 4], [], [], [6, 7], [6, 7, 8], [6, 7, 8, 9],
 [6, 7, 8, 9, 10]], [[], [], [], [], [7, 8, 4], [], [], [7, 8, 4, 7], [7, 8], [7
, 8, 9], [7, 8, 9, 10]], [[], [], [], [], [8, 4], [], [], [8, 4, 7], [8, 4, 7, 8
], [8, 9], [8, 9, 10]], [[], [], [], [], [], [], [], [], [], [], [9, 10]], [[],
[], [], [], [], [], [], [], [], []]]
```